P4 Expert Roundtable Series

April 28-29, 2020

Hosted by: ONF

Sponsored By

Google intel STORDIS The Open Networking Expert

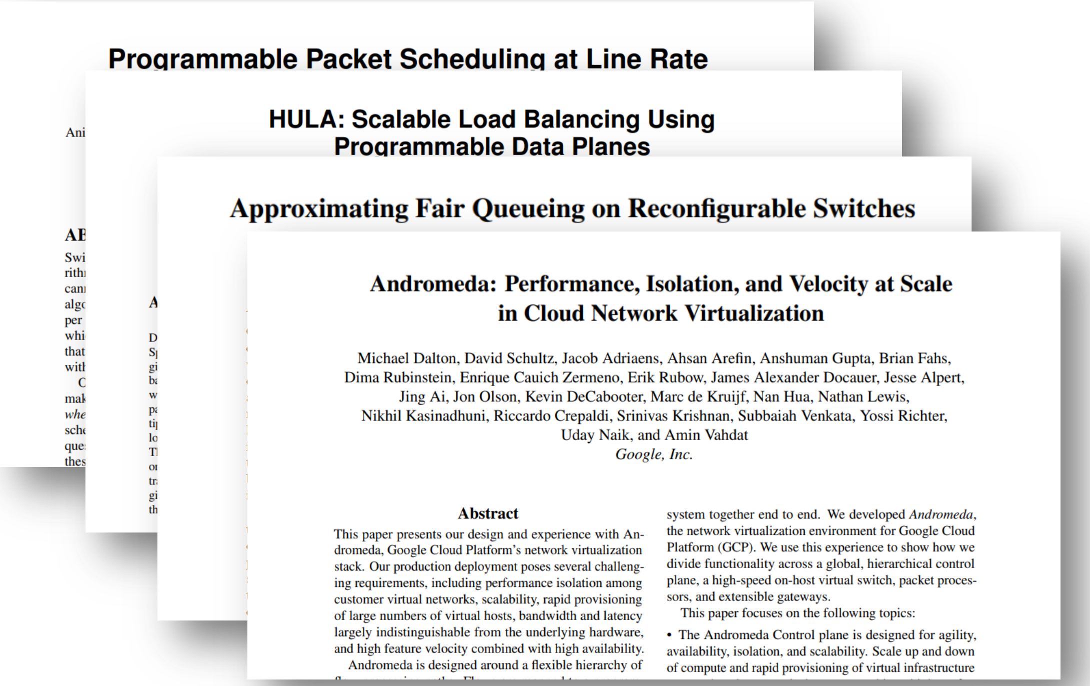
Mathematical Operations in Programmable Switches using TCAMs

Mojtaba Malekpourshahraki
Komal Shinde
Balajee Vamanan
Brent Stephens

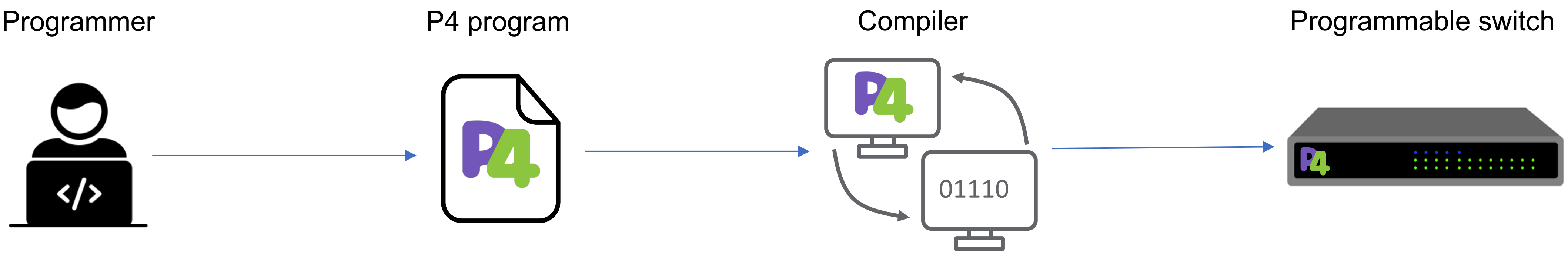
University of Illinois at Chicago

Programmable switches enable new switch functionality

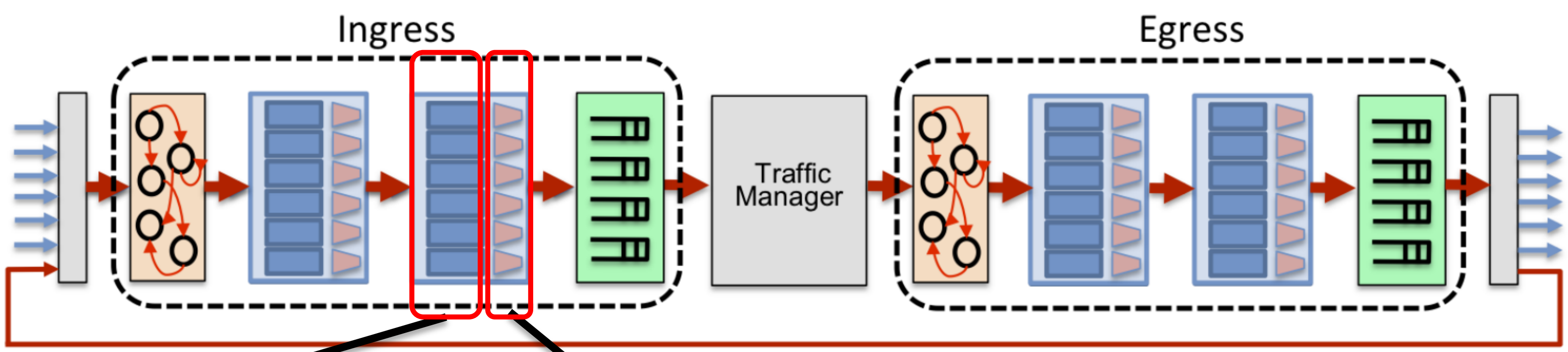
- Example use cases
    - Congestion control
    - In-network computation
    - Distributed consensus
    - Monitoring, and measurement
    - Load balancing



Today’s workflow for deploying switch programs

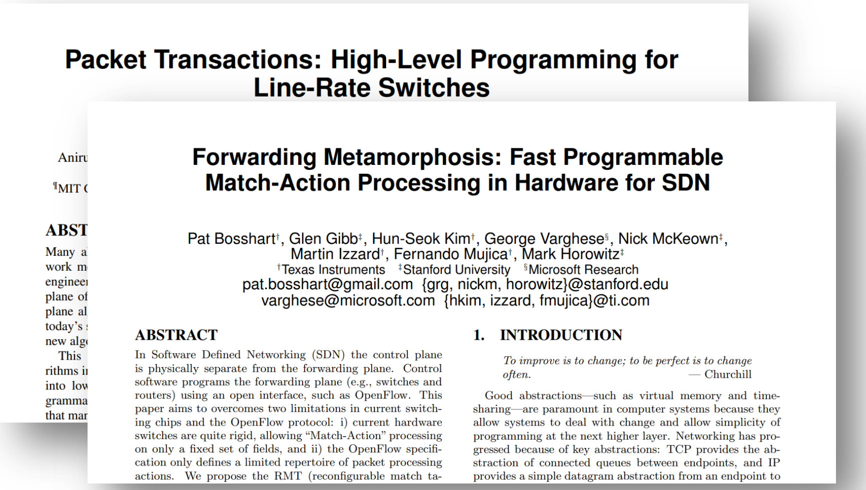


Architecture of programmable switches



Lookup tables
Compare input against the table and return the matching data.

ALUs
Process standard arithmetic operations, shift bit, header modification, hashing operation



Packet Transactions: High-Level Programming for Line-Rate Switches

Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Pat Bosshart[†], Glen Gibb[‡], Hun-Seok Kim[†], George Varghese[§], Nick McKeown[‡], Martin Izzard[†], Fernando Mujica[†], Mark Horowitz[‡]
[†]Texas Instruments [‡]Stanford University [§]Microsoft Research
pat.bosshart@gmail.com {grg, nickm, horowitz}@stanford.edu
varghese@microsoft.com {hkim, izzard, fmujica}@ti.com

ABSTRACT

In Software Defined Networking (SDN) the control plane is physically separate from the forwarding plane. Control software programs the forwarding plane (e.g., switches and routers) using an open interface, such as OpenFlow. This paper aims to overcomes two limitations in current switching chips and the OpenFlow protocol: i) current hardware switches are quite rigid, allowing "Match-Action" processing on only a fixed set of fields, and ii) the OpenFlow specification only defines a limited repertoire of packet processing actions. We propose the RMT (reconfigurable match ta-

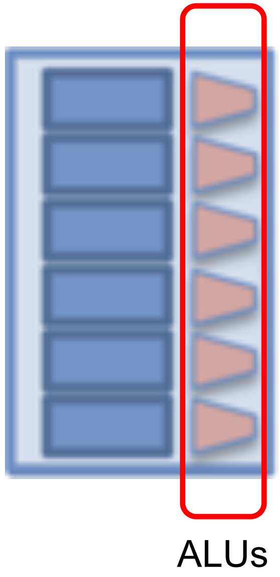
1. INTRODUCTION

To improve is to change; to be perfect is to change often. — Churchill

Good abstractions—such as virtual memory and time-sharing—are paramount in computer systems because they allow systems to deal with change and allow simplicity of programming at the next higher layer. Networking has progressed because of key abstractions: TCP provides the abstraction of connected queues between endpoints, and IP provides a simple datagram abstraction from an endpoint to

Current switches support only a limited set of mathematical operations

- Arithmetic logic unit (ALU) in programmable switches is limited



Supported operations

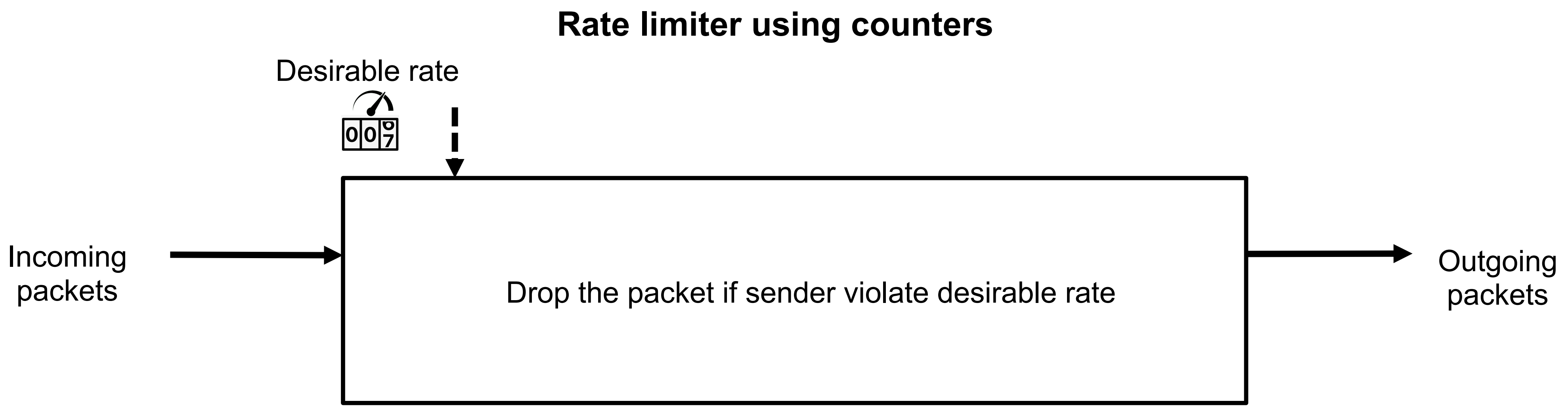
- Arithmetic operations: addition, subtraction
- Logical operations: left and right bit-shifts
- Header modification
- Hashing operation

Unsupported operations

- Multiplication
- Division

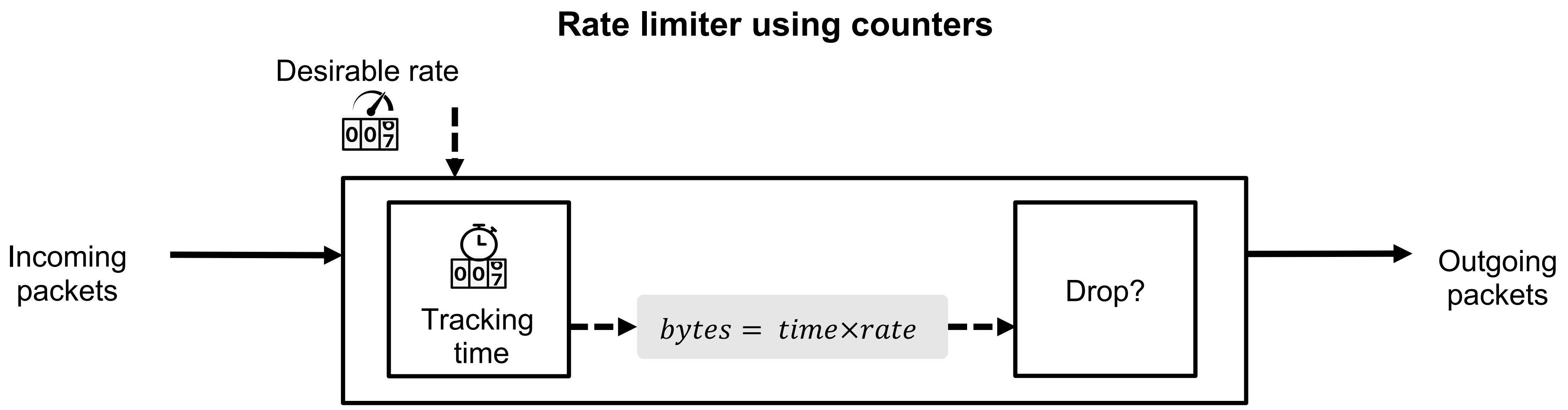
An example application on PISA

- Simple rate limiter on PISA via packet drops
  - Using counters to emulate a queue
  - Drop the packet if queue size exceeds drop threshold

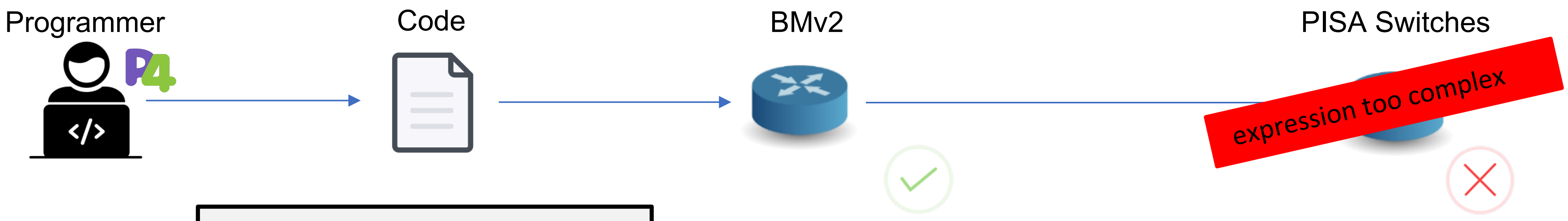


An example application on PISA

- Simple rate limiter on PISA via packet drops
  - Using counters to emulate a queue
  - Drop the packet if queue size exceeds drop threshold



Multiplication in ALUs

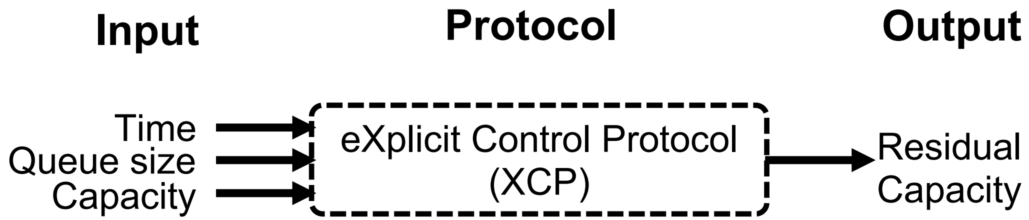



```
...
value = (bit<32>)time*rate;
```

ALUs on programmable switches do not support multiplication

Many applications need mathematical operations

- Example use cases
  - **Congestion control**
  - In-network computation
  - Distributed consensus
  - Monitoring, and measurement
  - Load balancing



- Four floating-point multiplications

- Seven multiplications/divisions

Many key applications require multiplication

How to provide multiplication in PISA?

Exact Multiplication	Approximate Multiplication
✗ **Multiplication with addition and shift-bits** • Exact result • Prohibitive, requires too many stages	✗ **Approximate multiplication using left shift** • Low overhead • Large error
✗ **Exact match lookup table** • Exact result • Prohibitive, requires large memory	✓ **Approximate using lookup table** • Low overhead

Can we emulate multiplication using
reasonably sized lookup tables and provide error bounds?

Contributions

- Propose an approximation approach for multiplication
  - Bounded maximum error
  - Limited number of table entries

- Present analysis of maximum error and TCAM usage
  - Derive the optimal number of entries for a *given* error

Outline

- Introduction
  - Use cases for programmable switches
  - Key limitation of existing switches
  - A motivating example
  - Our contributions
- Design
- Evaluation
- Conclusion

Naïve solution to populate lookup table

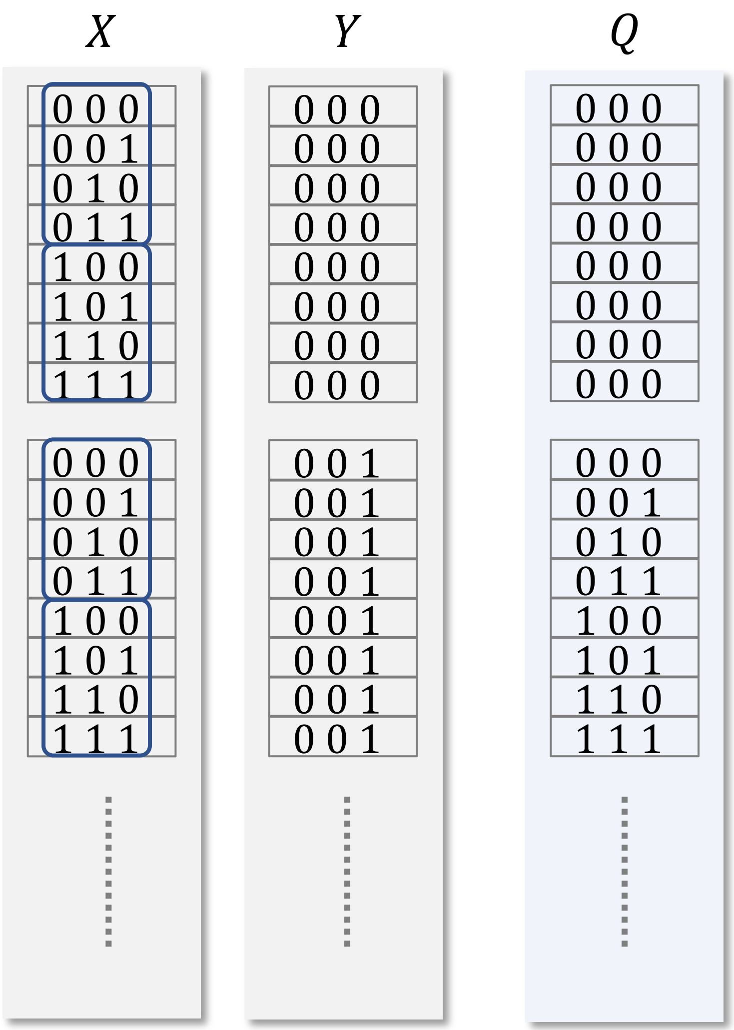
- Naïve approach to calculate $X \times Y = Q$
  - Requires large number of entries

$2^6 = 64$

X	Y	Q
0 0 0	0 0 0	0 0 0
0 0 1	0 0 0	0 0 0
0 1 0	0 0 0	0 0 0
0 1 1	0 0 0	0 0 0
1 0 0	0 0 0	0 0 0
1 0 1	0 0 0	0 0 0
1 1 0	0 0 0	0 0 0
1 1 1	0 0 0	0 0 0
0 0 0	0 0 1	0 0 0
0 0 1	0 0 1	0 0 1
0 1 0	0 0 1	0 1 0
0 1 1	0 0 1	0 1 1
1 0 0	0 0 1	1 0 0
1 0 1	0 0 1	1 0 1
1 1 0	0 0 1	1 1 0
1 1 1	0 0 1	1 1 1
⋮	⋮	⋮

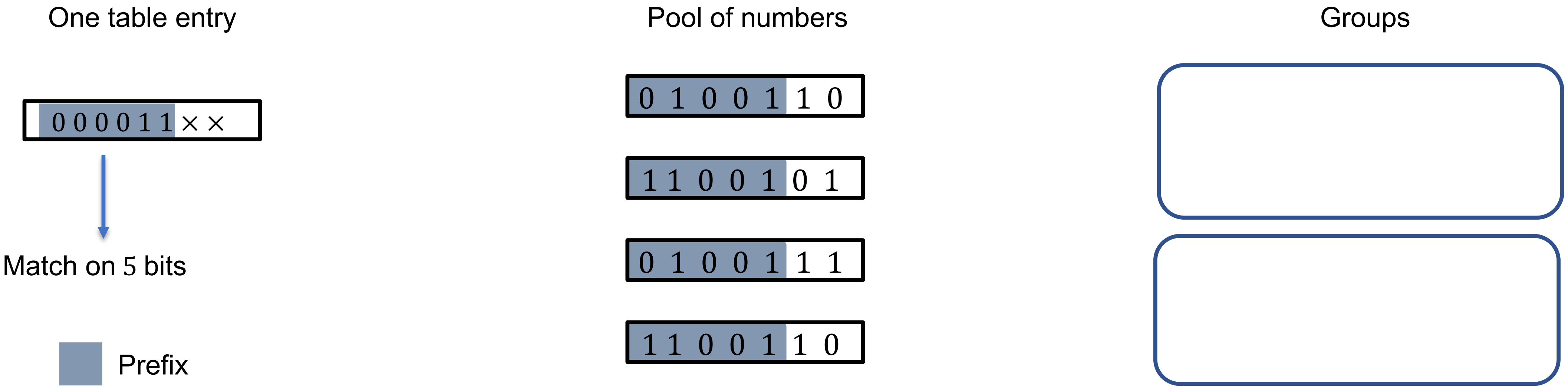
Our solution: reducing the table size

- Grouping numbers such that
  - Numbers in the group are close
  - Each group has a *group head*
  - Using group head to calculate the product



How to group? #1 Rounding

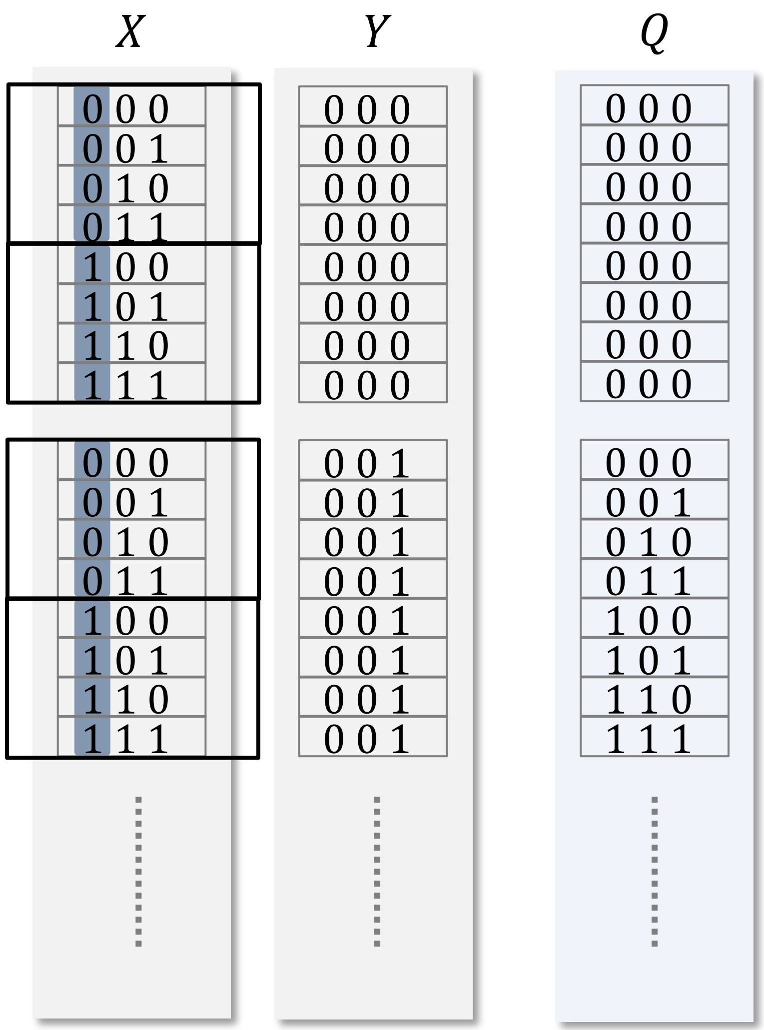
- Group numbers based on
  - n most significant bits



How to group? #1 Rounding

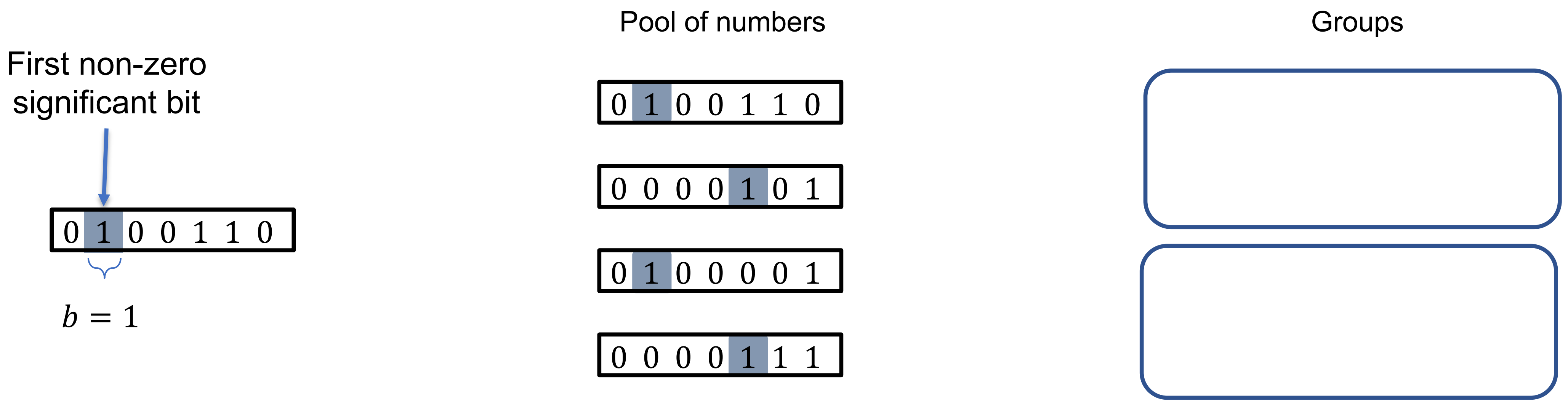
- Match on a *fixed* number of most significant bits
  - This example matches on one bit

- Pros/Cons
  - Simple
  - Large error
  - Large number of entries



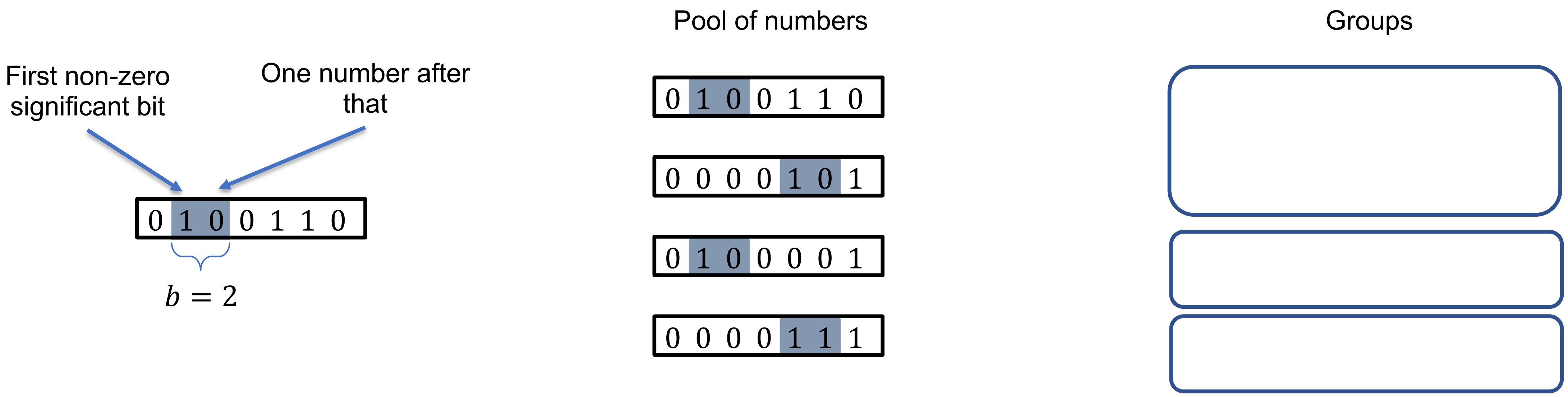
How to group? #2 Adaptive precision

- Group numbers based on
  - The *first* non-zero significant bit:



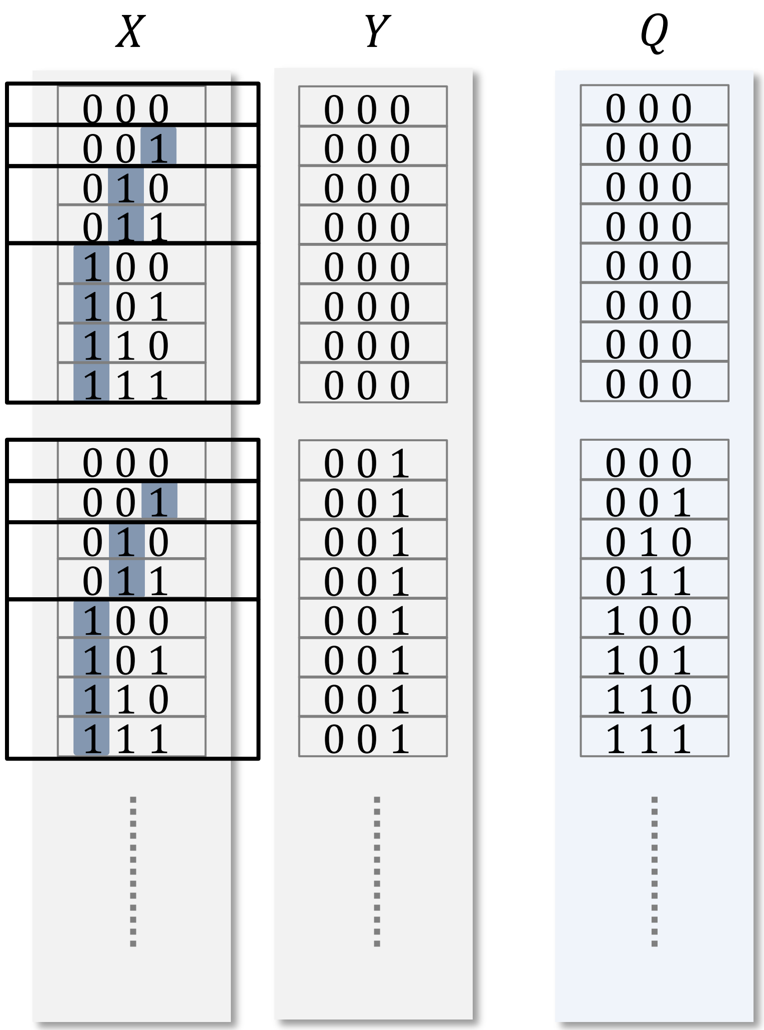
How to group? #2 Adaptive precision

- Group numbers based on
  - ***Two digits*** after first non-zero significant bit $(b = 2)$



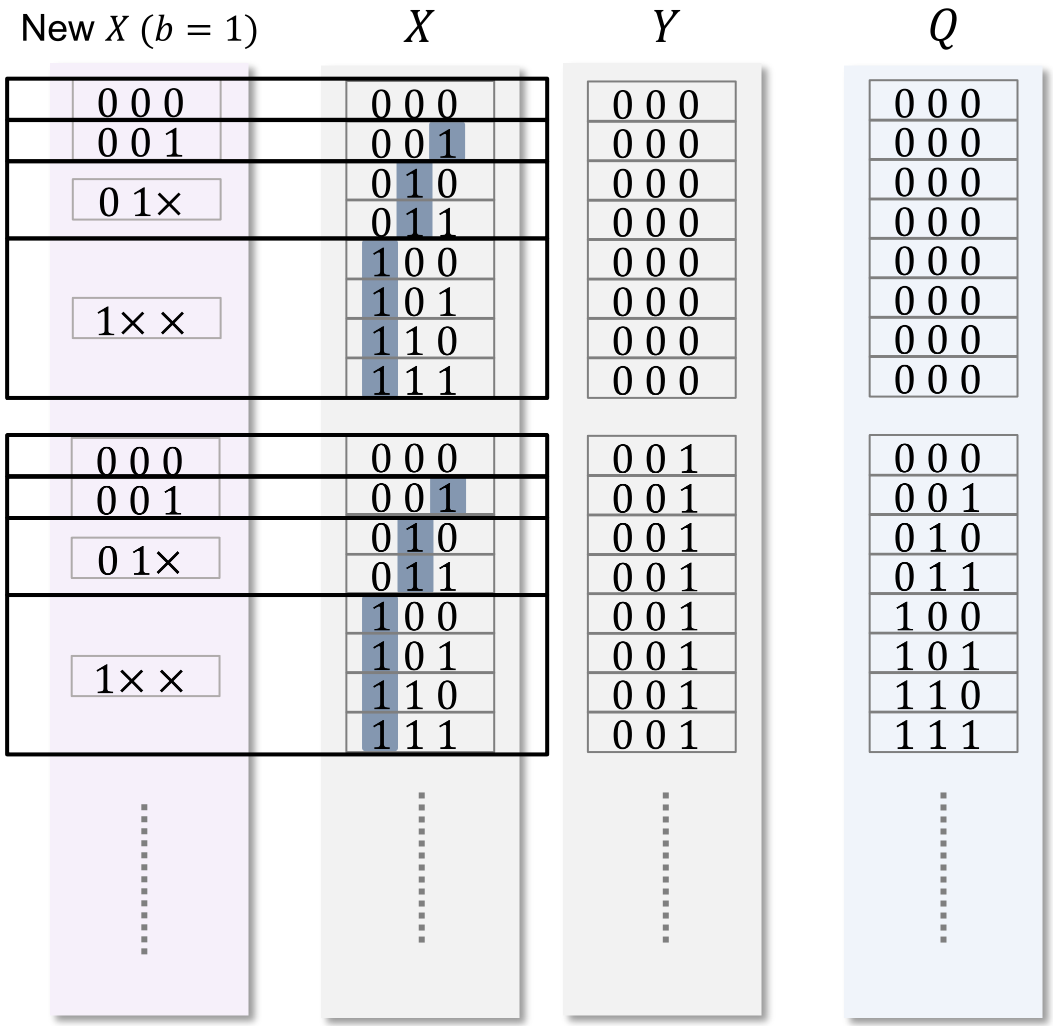
How to group? #2 Adaptive precision

- Use *varying* number of matching bits per group
  - Grouping is based on first non-zero most significant digit ($b = 1$)
- Pros/Cons
  - Bounded error
  - Limited memory usage
  - Different precision for different groups



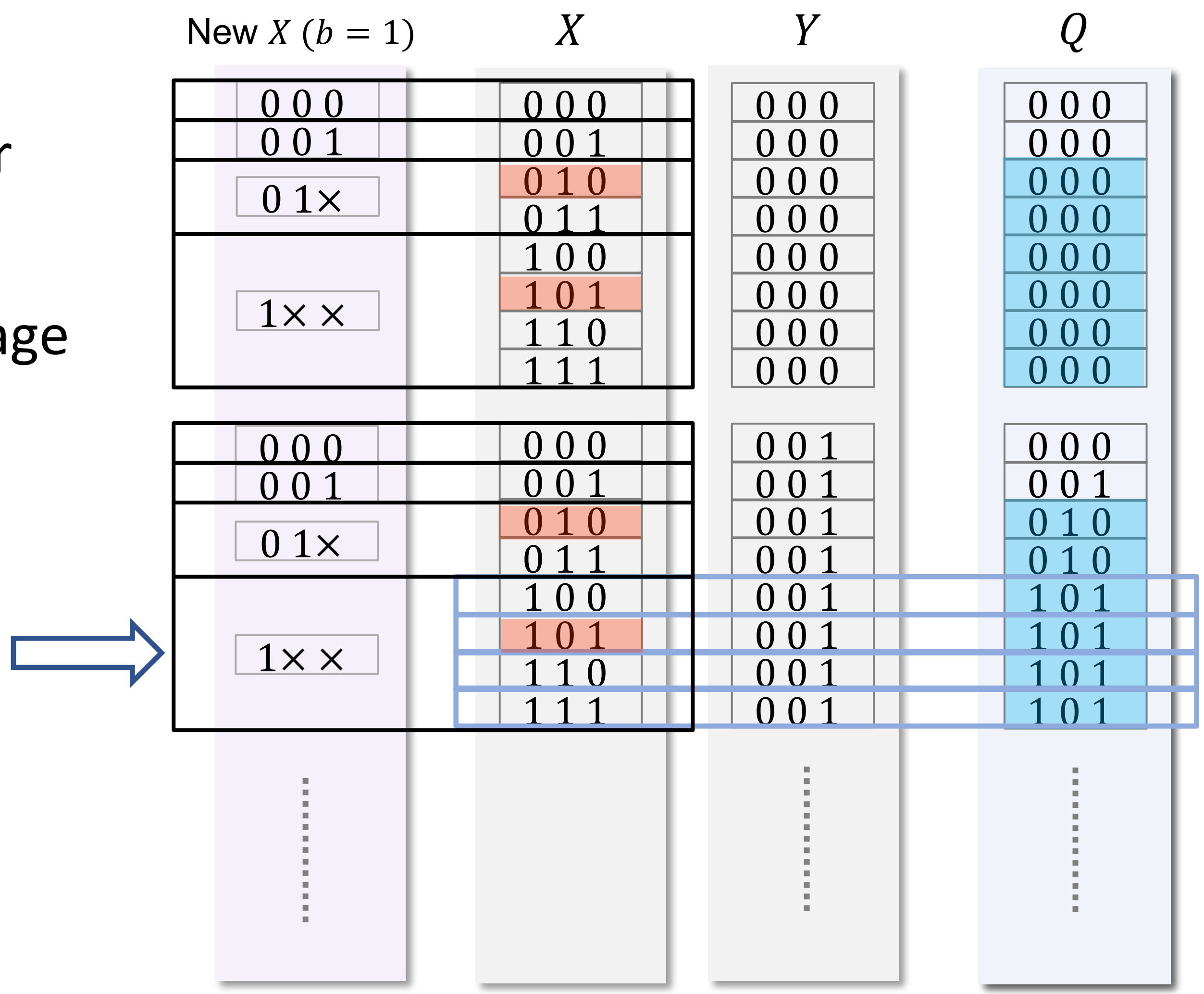
Using LPM in grouping

- LPM allows representing groups in TCAMs
- Implement grouping with LPM for one bit (b=1)



Choosing group head

- Choose group head to minimize error
- Idea: choose a number close to average
  - why: average is known to minimize error

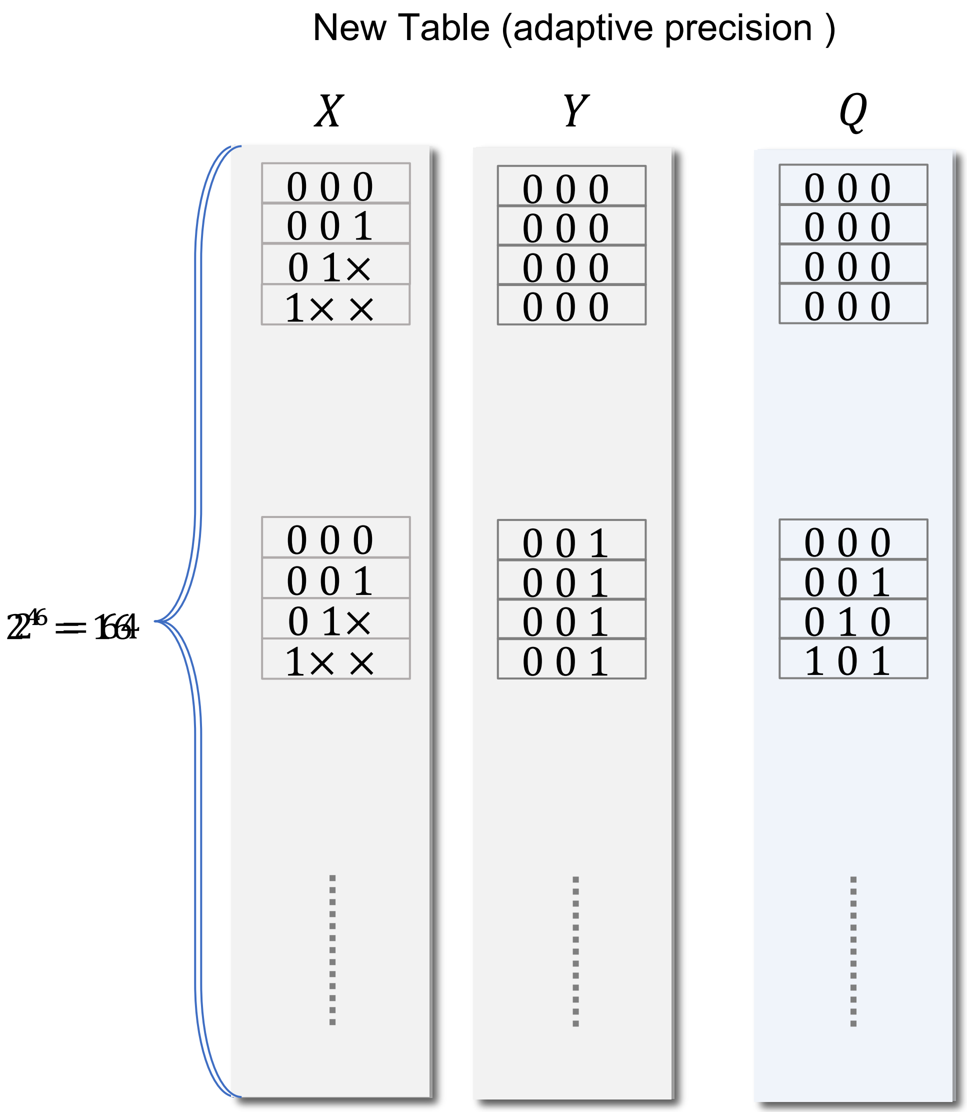


More information in paper

Smaller table size using LPM

- The number of table entries in this approach is 75% less than naive approach

- Error increases when the value Increases



Analysis of memory overhead and error

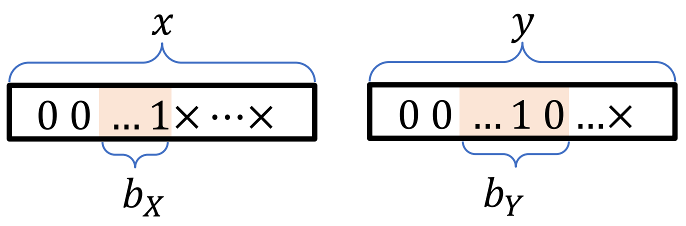
Maximum approximation error

$$Error = \frac{(2^x+1)\left(\frac{2^y}{2^{b_Y}}\right)+(2^y+1)\left(\frac{2^x}{2^{b_X}}\right)-1}{2^x 2^y}$$

Number of entries

$$\begin{cases} e_X = (x-b_X+2)2^{b_X-1} \\ e_Y = (y-b_Y+2)2^{b_Y-1} \end{cases}$$

$$Number\ of\ entries = e_X e_Y \frac{2^q(1+\ln\frac{2^x 2^y}{2^q})}{2^x 2^y}$$



More information in paper

Outline

- Introduction
  - Use cases for programmable switches
  - Key limitation of existing switches
  - A motivating example
  - Our contributions
- Design
- Evaluation
- Conclusion

Methodology

- All numbers have same number of bits
  - 32 bits for all numbers

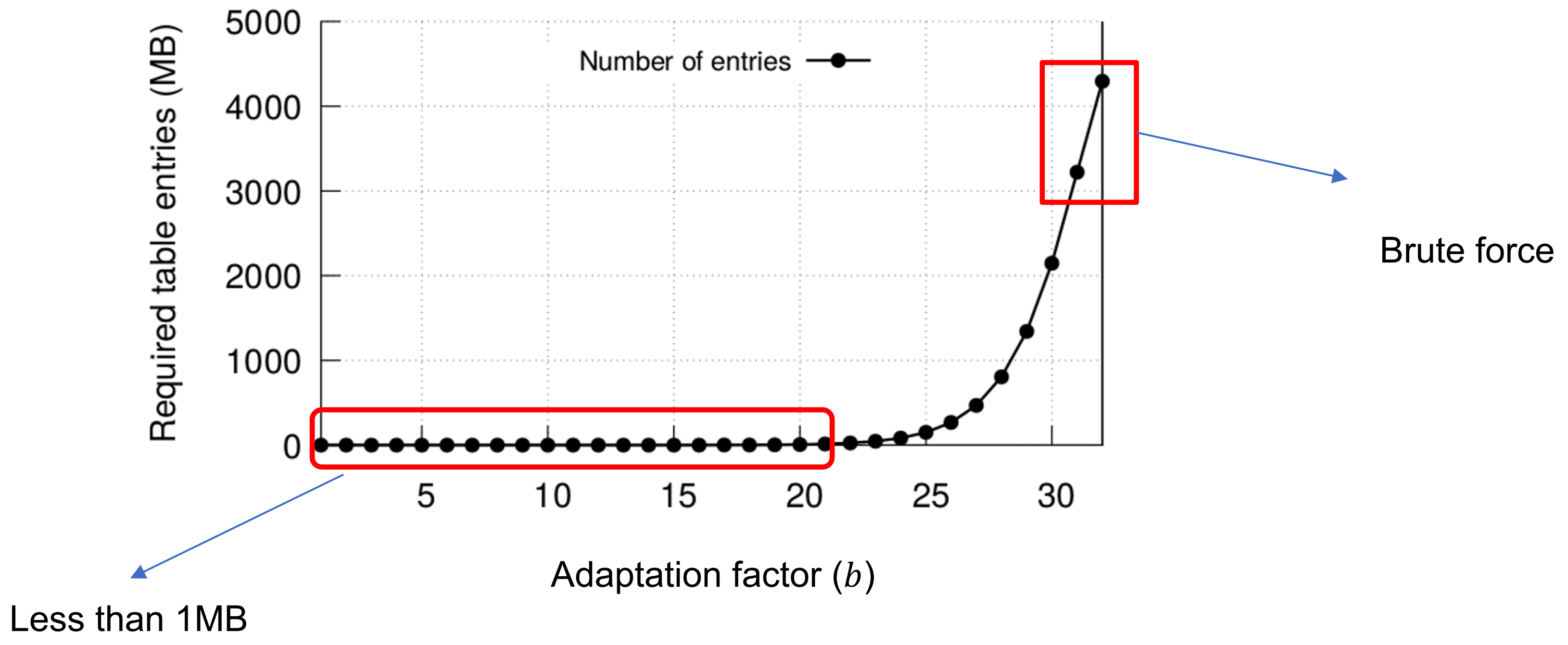
	X	$\times$	Y	$=$	Q
Numbers	X		Y		Q
Numbers of bits	32		32		32

  - Same error adaptation factor for X and Y ($b_x = b_y$)

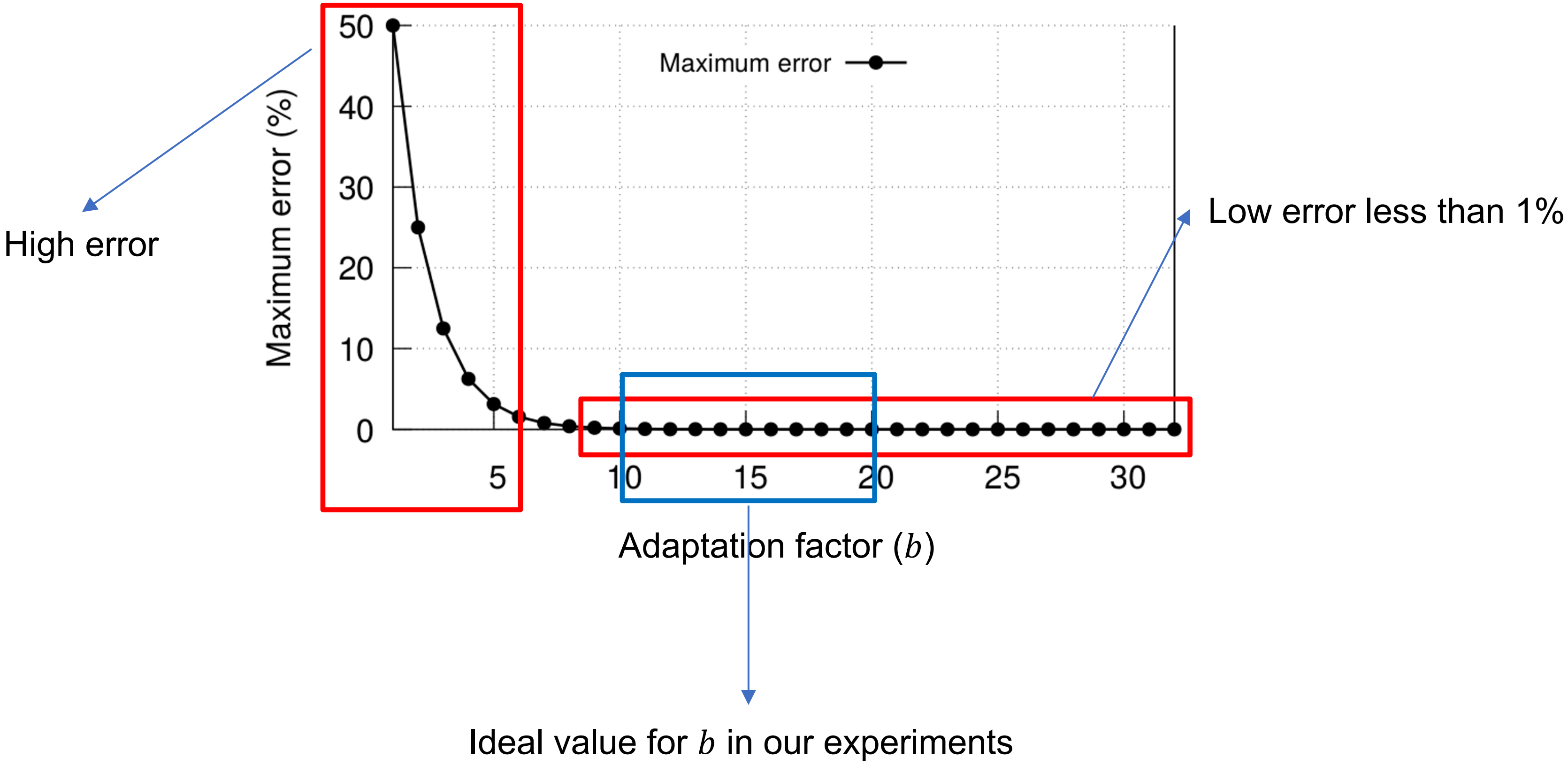
- Evaluation Goal:

How error and number of entries changes when b changes?

Number of entries (memory overhead)

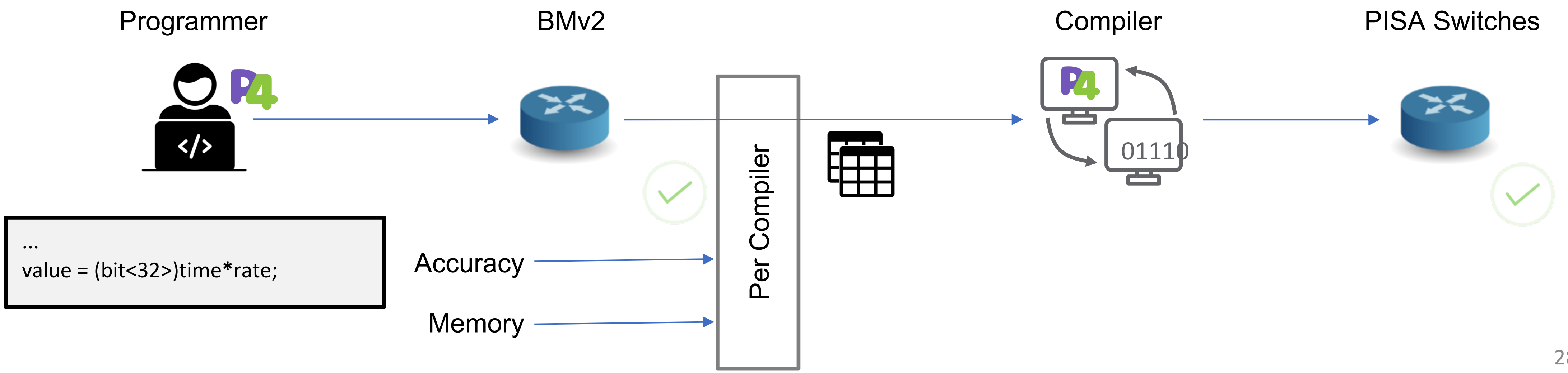


Accuracy (error)



Conclusion and future work

- We designed an approximate approach to support multiplication using lookup tables
- We are working toward Implementing a P4 pre-compiler that
  - Converts a P4 program with multiplication → P4 program with lookup tables
    - Can achieve a suitable trade-off between table size and accuracy (error)



Thank You

mmalek3@uic.edu

www.cs.uic.edu/~mmalekpo